# Special Timing BR Considerations

**Wesley Ketchum**

**(and artdaq team)**

## InhibitMaster Ideas

**And discussion of plans…?**

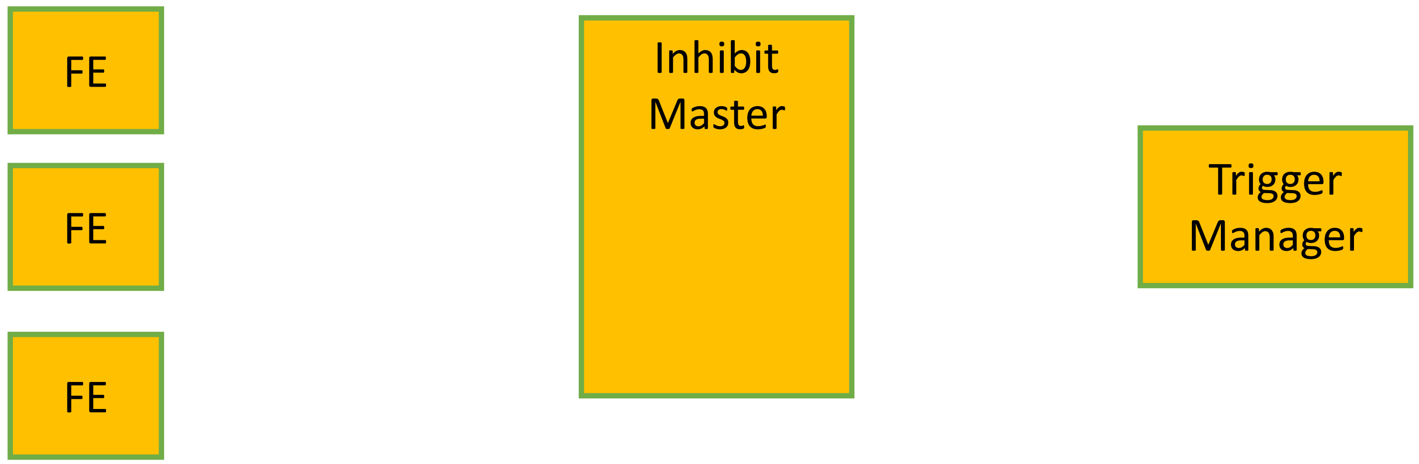
## Requirements

- Want software control to issue inhibit on trigger hardware
- Multiple contributors that can generate an inhibit
    - *BoardReaders or other software entities (like artdaq processes?)*
- Global inhibit state calculated and available to trigger
- Fast response (milliseconds latency?)
- Simple and comprehensive monitoring
    - *Necessary for easy debugging/understanding of inhibits*

## IMPLEMENTATION

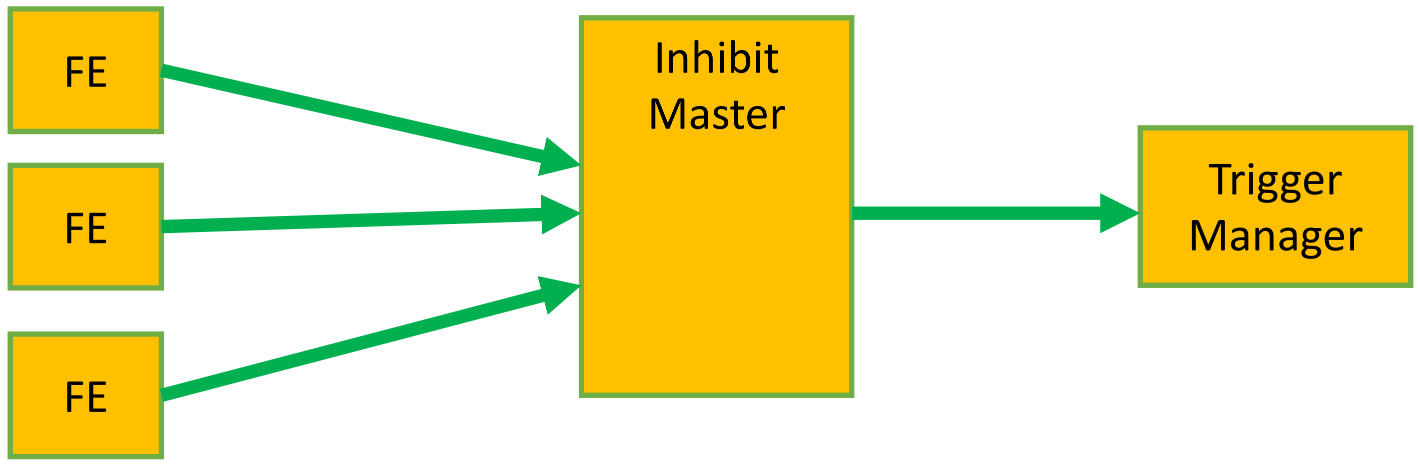
- Python-based demonstrator using ZeroMQ
  - *With C-based frontend demonstrator as well*
- Frontends publish inhibit status (GOOD/BAD)
  - *PUB-SUB model, so no need to specify subscribers*
- InhibitMaster subscribes to messages and determines global state
  - *FrontEnds to subscribe currently must be known at configuration*
- InhibitMaster publishes global status (GOOD/BAD)
- Trigger software subscribes to global status and acts on state
- Monitoring program and subscribe to all messages and verify/log state changes independently

## MESSAGE/STATE FLOW DIAGRAM



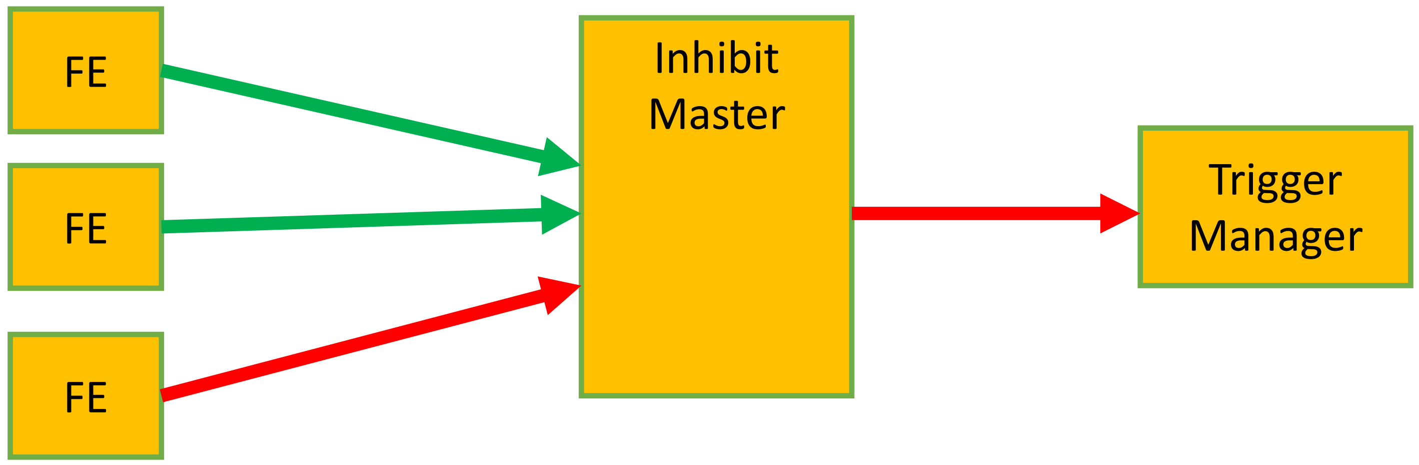
## MESSAGE/STATE FLOW DIAGRAM

- InhibitMaster receives three “GOOD” messages, and passes along the goodness



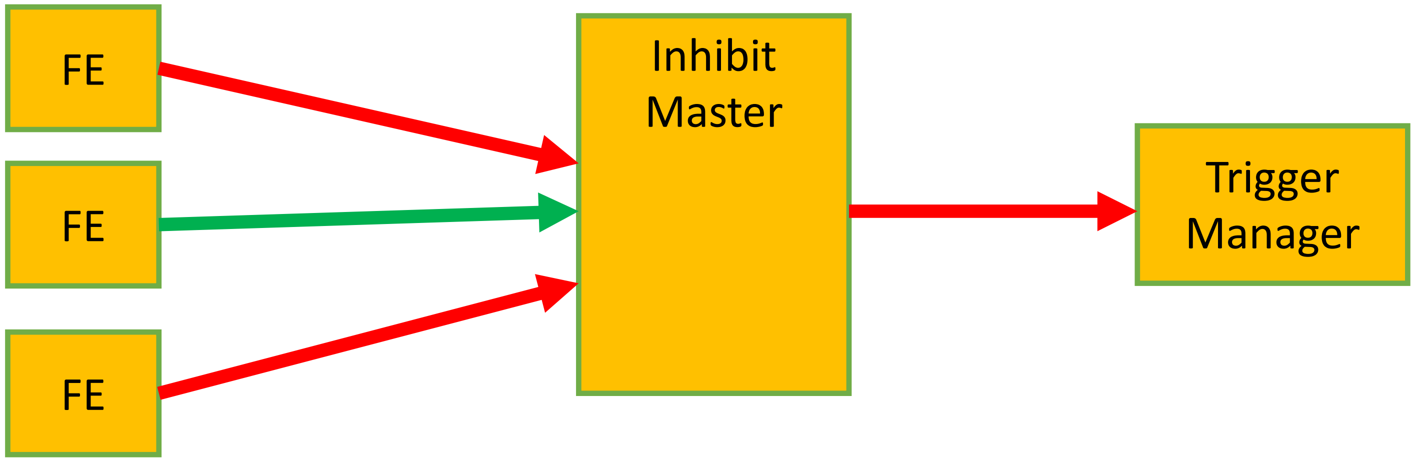
## MESSAGE/STATE FLOW DIAGRAM

- Upon seeing state change in one FE, IM updates glboal state



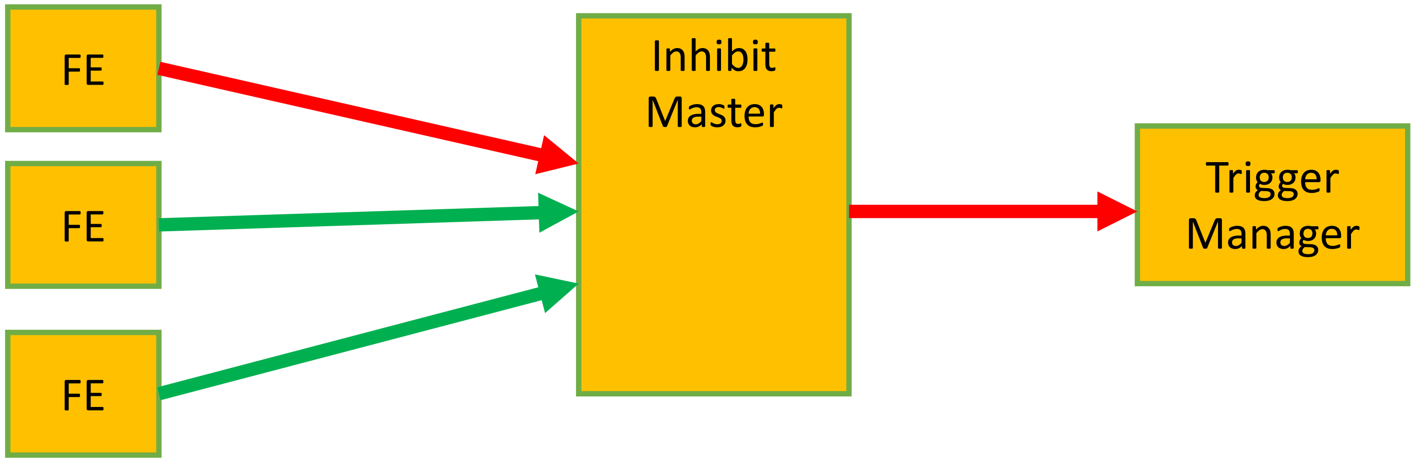
## MESSAGE/STATE FLOW DIAGRAM

- Another FE reports inhibit, and IM recalculates state (still bad), and keeps sending inhibit



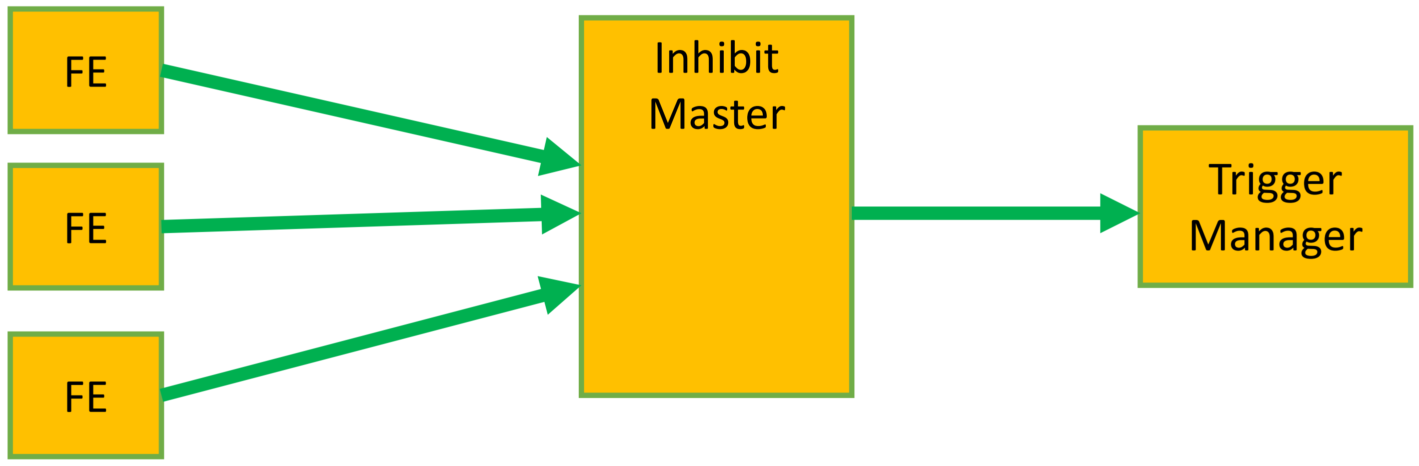
## Message/State Flow Diagram

- Initial “bad” FE returns to good. State change → IM recalculates state, but still bad



## Message/State Flow Diagram

- Upon FE changing state, IM recalculates and releases the global inhibit



## ADDITIONAL DETAILS

- InhibitMaster preferentially looks at state changes
- Can be configured to have no queue/memory to go through per FE
  - *That is, you always read the current/most recent state*
- FrontEnds and InhibitMaster can broadcast messages, so everything also available to monitoring programs
  - *Check consistency, see which FrontEnd is causing inhibit, etc.*
- Each FrontEnd should report "GOOD/BAD" status to enable a stop
  - *RunControl can be an additional FrontEnd → this is a method for enacting enable/disable trigger transition*
  - *artdaq processes could allow for reporting buffers being full (in addition to providing backpressure) → prevent backpressure buildup*

# Plans for implementation

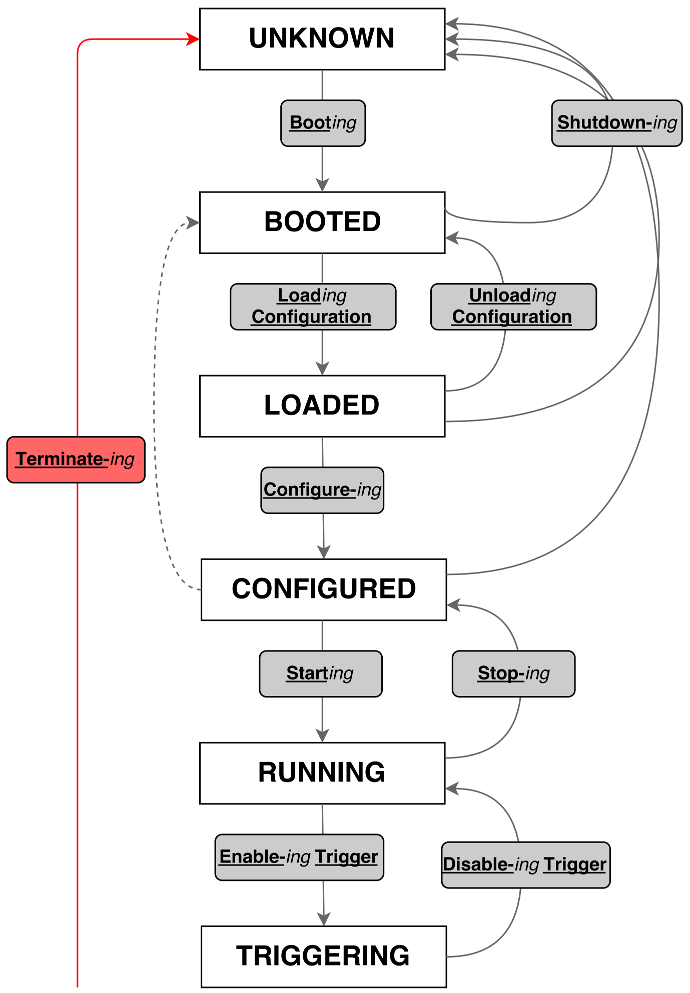
- Can push to implement ZeroMQ-based version in ProtoDUNE when we are ready
  - *Add starting/stopping InhibitMaster program inside DAQInterface*
    - Need to work out configuration of messaging ports
  - *Need to develop monitoring program to send data to DIM*
    - $\rightarrow$ state monitoring in JCOP more easily
  - *Would limit this to FrontEnd status for now in FragmentGenerators*
    - Not implement from artdaq queues
  - *TimingFragmentGenerator should be ready to act on messages(?)*
- Based on ProtoDUNE experience, outline design plan for integration in future version of artdaq (if desired)

## Trigger Configuration

**Discussion**

# State Machine

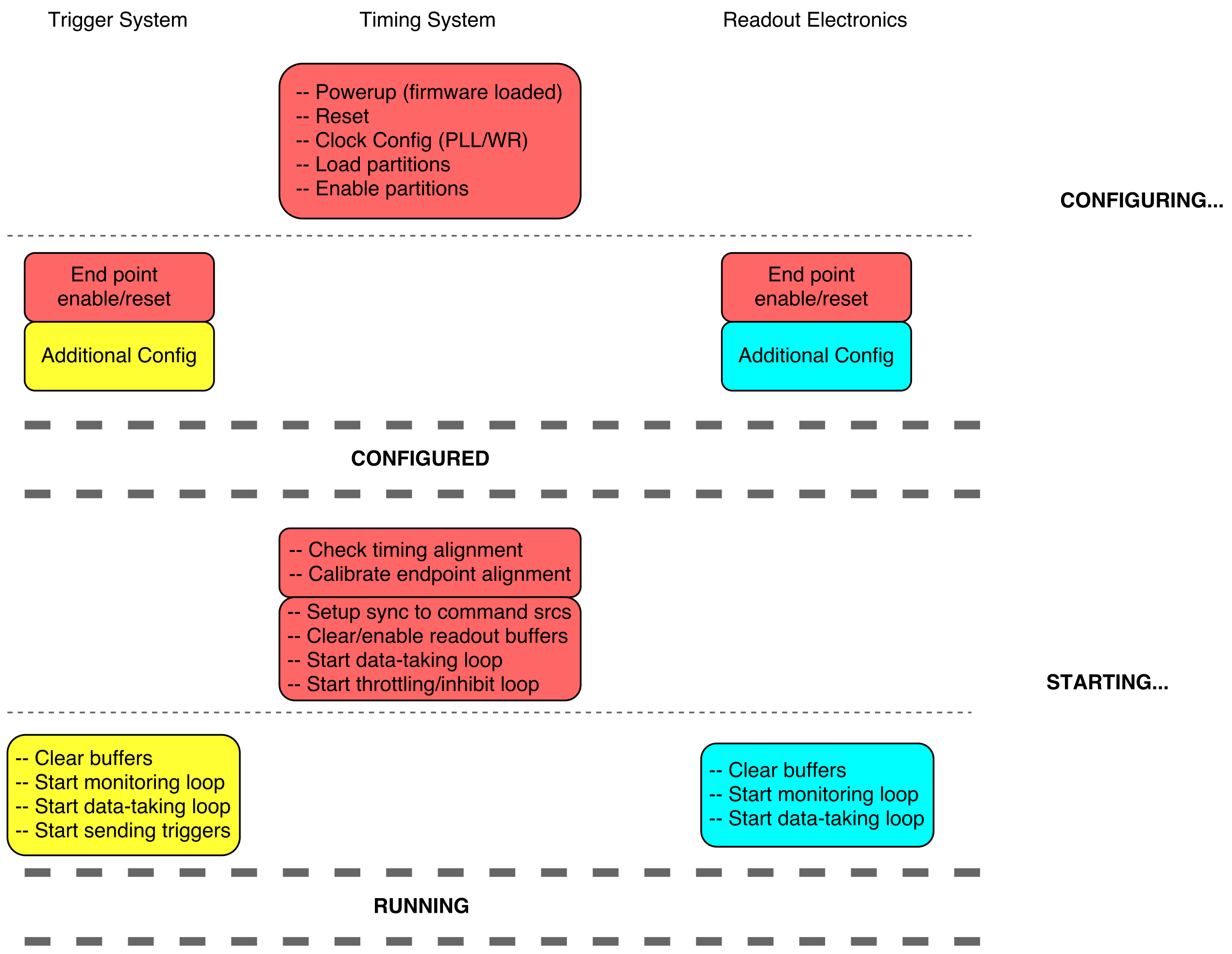
- From artdaq process view…
  - *Processes started at BOOT*
  - *Configuration extracted in "Load"*
  - *"Configure", "Start", and "Stop" transitions match*
    - Fragment Generator is constructed at configure
  - *No meaning in artdaq for "Enable"/"Disable" triggers*
    - Running allows all data flow



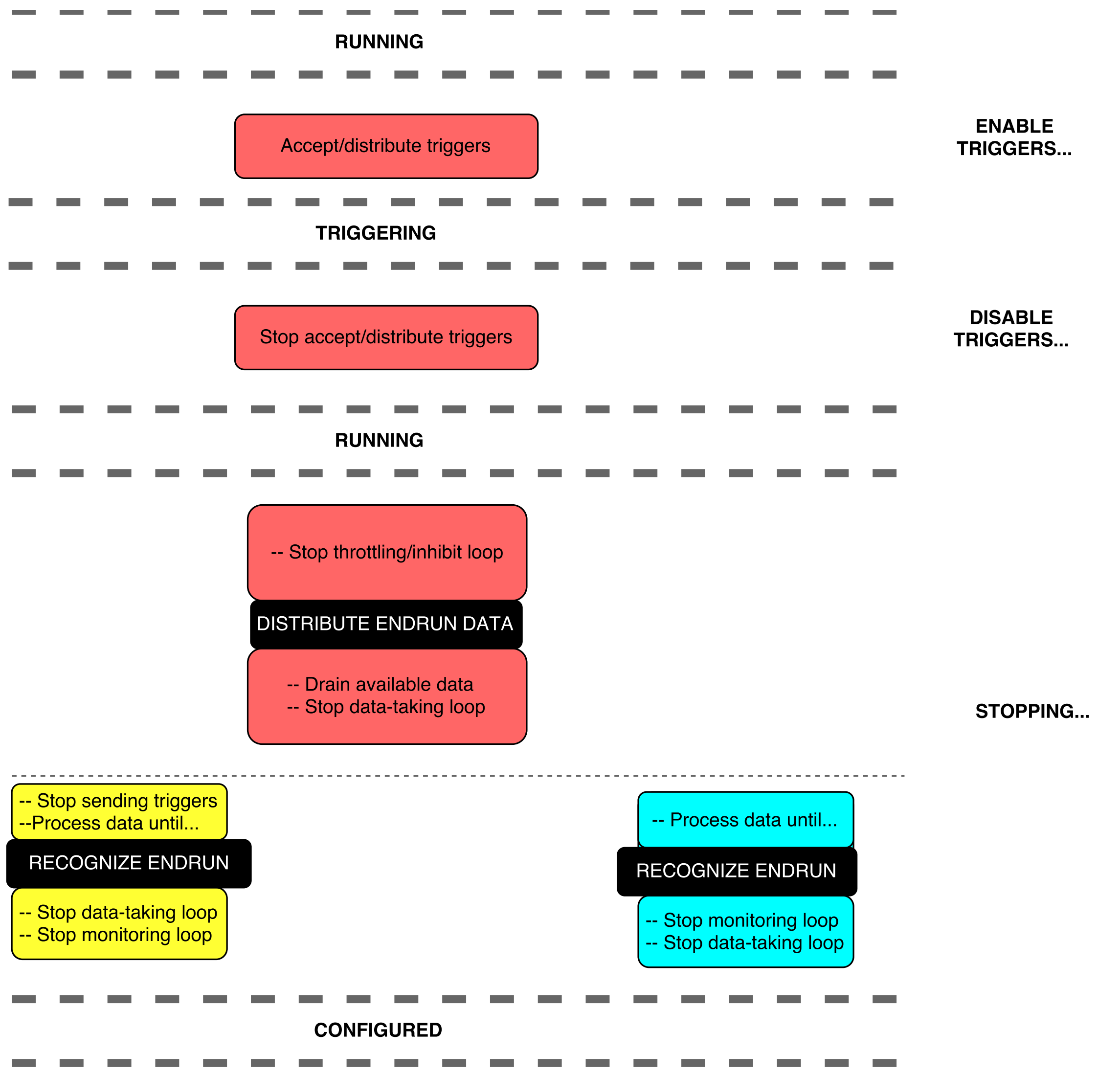
## REQUIREMENTS FOR TRIGGER* CONFIG

- These are my best understanding …
  - *****This is all on the timing system*
  - *We would like ability to stop data-taking, adjust *some* of the trigger configuration, and restart*
    - Enable/disable of bits, rates, and prescales?
  - *We want to do this without needing to reconfigure other parts of the system*
    - Artdaq processes stay up
    - Hardware outside of timing system doesn’t go through reconfiguration

# REVIEWING ORDER OF OPERATIONS (1)



# Reviewing order of operations (2)



## OPTIONS

1. Bring timing BR through a load/configure transition? No.
   - *Order of configure operations means completely resetting timing requires reconfiguration of endpoints*
2. Add (re)configuration of trigger conditions as part of Start transition? Seems possible.
   - *Need mechanism for passing in trigger configuration at start transition*
   - *Trigger config is static per run → add to RunHistory DB*
   - *Requires stopping runs not take forever with timeouts so we can quickly restart*
3. Add (re)configuration of trigger conditions as part of "Enable" transition? Seems possible.
   - *If enable/disable transition implemented via InhibitMaster, need mechanism for passing in trigger configuration when we see global state switch*
     - Completely internal to timing system
   - *Trigger config not static per run → need to add way to know config in data → probably metadata in timing fragment (per event!)*

## PROPOSAL

- I like option 2 because it makes me feel good *and* seems easiest to do and keep cleaner design elsewhere
  - *”A run is a period of time in which data was taken with a consistent configuration.”*
  - *All configuration is lookupable by run in same way*
  - *There’s no potential mixing of data taken from different configuration conditions*
    - Less important for data quality, but perhaps more important for downstream OM of trigger rates?
- Mechanism to pass in trigger config? Can we do this in the config DB?
  - *Can we have a special “ActiveTriggerConfig” configuration, which can point to labeled/versioning trigger configs?*
    - In start transition, timing system ALWAYS looks up “ActiveTriggerConfig” and applies it
    - User actions to add new trigger configs and switch ActiveTriggerConfig
    - ActiveTriggerConfig pulled down and included as part of RunHistory DB (by DAQInterface? on start transition?)